Licenciatura Engenharia Informática e Multimédia

Modelação e Programação – MP

Relatório Trabalho Prático 1

Docente Pedro Fazenda

Trabalho realizado por:

Fábio Dias, nº 42921

Índice

Índice **1**

1. pack1Revisoes **2**

1.1. P01CheckPrime 2

1.2. P02FourInaRow 5

1.3. P03WorkWithStrings 13

2. pack2Livros **17**

2.1. Livro 17

2.2. Coleccao 25

3. pack3Coleccoes **35**

3.1. Coleccao 35

1. pack1Revisoes

1.1. P01CheckPrime

Para fazer com que o programa possua um *loop*, dei uso a um ciclo *while* como uma variável booleana *isExecuting*. Esta será mudado caso o utilizador insira um "0". Tento converter o input recebido para um *int* mas encapsulo esta transformação num *try-catch*. Se não conseguir converter para *int*, uma *flag* entra acção e termina o *loop* por ali.

De seguida, verifico se o input é "0", para finalizar o programa, ou caso contrário, para pedir um novo *input* ao utilizador enquanto o informo que o *input* recebido não é válido.

Finalmente, chamo a função *isPrime* que recebe o *input* e me devolve um *boolean*: *true* se o número for primo, *false* caso contrário.

Neste método, começo por verificar se o número é 1, sendo este primo, e vou incrementando uma variável auxiliar e obter o resto da divisão por esse. Caso seja 0, o número não é primo, caso não seja, é.

Código:

```
package tp1.pack1Revisoes;

import java.util.Scanner;

public class P01CheckPrime {

    /**
     * Corre o programa, pedindo ao jogador um número inteiro e validando o
seu input.
     */
    public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);
      boolean isExecuting = true;

      int number = -1;

      while(isExecuting)
      {
            boolean isString = false;

            System.out.println("Introduza um número inteiro positivo: ");
            String inputReceived = scanner.nextLine();



            try
            {
                  number = Integer.parseInt(inputReceived);
            }
            catch (Exception e) {
                  System.out.println("Input recebido é inválido.");
                  System.out.println();
                  isString = true;
            }
```

```
            if(!isString)
            {
                  if(number == 0)
                  {
                        isExecuting = false;
                        System.out.println("Programa Terminado");
                  }
                  else if(number < 0)
                  {
                        System.out.println("Input recebido é inválido. O
número tem de ser positivo.");
                        System.out.println();
                  }
                  else
                  {
                        System.out.print("O número " + number);

                        if(isPrime(number))
                        {
                              System.out.print(" é");

                        }
                        else
                        {
                              System.out.print(" não é");

                        }
                        System.out.println(" primo");
                        System.out.println();
                  }
            }
      }
      scanner.close();
   }
```

```java
/**
     * Avalia se o número introduzido pelo utilizador é, ou não, primo.
     *
     * @param number O número a ser avaliado
     */
    public static boolean isPrime(int number) {

      int i = 2;
      boolean isPrime = true;

      if(number == 1)
      {
            return true;
      }

            while(number / 2 >= i)
            {
                  if(number % i == 0)
                  {
                        isPrime = false;
                        break;
                  }

                  i++;
            }

      if(isPrime)
      {
            return true;
      }

        return false;
    }
}
```

1.2. Po2FourInaRow

Para o método *board*, que recebe como argumento um *array* bidimensional de caracteres, este é percorrido e caso o *char* seja vazio, que tem o código "\u0000" é imprimido um 0, caso seja um caracter correspondente a um dos jogadores, este é impresso.

No método *play*, é recebido o caractere representativo do jogador, o tabuleiro e uma instância da classe Scanner. A intenção desta função é realizar uma jogada. É realizado um *loop* até receber um input válido, que será o número da coluna pretendida, de 1 a 6. Caso não seja, uma mensagem informará o jogador que o input não é válido. É subtraída uma unidade ao valor recebido, de forma a ficar com o valor entre os índices válidos do *array*, e são verificadas as linhas, de baixo para cima, até encontrar o caractere vazio. Caso não o encontre, a coluna encontra-se cheia de caracteres dos jogadores e, consequentemente, é pedido novamente um input do jogador.

Na função *existsFreePlaces*, cujo argumento é o tabuleiro, este é percorrido e, caso exista um caractere vazio, é devolvido *true*, indicando que existem espaços vazios. Caso não haja, é devolvido um *false*.

Chegamos ao método que verifica se na última jogada efetuada, saiu um vencedor. Este recebe como argumentos um número inteiro, que é a coluna onde foi efectuada a última jogada, e o tabuleiro. Nessa coluna, é procurada a última peça jogada, ou seja, a última diferente de vazio. Após a encontrar, verifica-se se na mesma linha existem quatro peças do mesmo tipo seguidas. Caso esta situação não seja encontrada, é verificado na coluna mas apenas para baixo. Sempre que não for encontrado um vencedor, é passada para o método de procura seguinte. Após a procura na coluna, são avaliadas as diagonais. Para tal, é encontrado o lado mais esquerdo possível e o mais acima possível, é percorrida a diagonal para a direita, no sentido inferior. Fazemos o mesmo no sentido superior direito, para o sentido inferior esquerdo.

Caso encontremos quatro seguidas em qualquer uma destas situações, é retornado o valor *true*, que significa que existe um vencedor. Caso não seja encontrado, é devolvido um *false*, indicando que ainda não há vencedor.

Finalmente, no método *main*, é onde ocorre o *game loop*. É instanciado o tabuleiro, damos-lhe *display*, e, enquanto existirem espaços vazios, chamamos o método *play* para o jogador "A", de seguida verificamos se este ganhou o jogo, se existem espaços vazios e repete para o jogador "B".

Código:

```
package tp1.pack1Revisoes;

import java.util.Scanner;
```

```
public class P02FourInaRow {

    /**
     * Shows (prints) the board on the console
     *
     * @param board The board
     */
    private static void showboard(char[][] board) {

      System.out.println("+-------------+");

      for(int rows = 0; rows < board.length; rows++)
      {
            System.out.print("| ");

            for(int cols = 0; cols < board[rows].length; cols++)
            {
                  if(board[rows][cols] == '\u0000')
                  {
                        System.out.print('0' + " ");
                  }
                  else
                  {
                        System.out.print(board[rows][cols] + " ");
                  }
            }

            System.out.println("|");
      }

      System.out.println("+-------------+");
      System.out.println();
    }

    /**
       * Puts one piece for the received player. First asks the user to
choose one
       * column, then validates it and repeat it until a valid column is
chosen.
     * Finally, puts the player character on top of selected column.
     *
      * @param player   The player: 'A' or 'B'. Put this character on the
board
     * @param board    The board
     * @param keyboard The keyboard Scanner
     * @return The column selected by the user.
     */
    private static int play(char player, char[][] board, Scanner keyboard) {

      boolean isExecuting = true;
      int number = -1;

      while(isExecuting)
      {
            boolean isString = false;

            System.out.print("Choose a column (Player " + player + "): ");
            String inputReceived = keyboard.next();

            try
            {
```

```
                        number = Integer.parseInt(inputReceived);
                }
                catch (Exception e) {
                        System.out.println("Invalid Input.");
                        isString = true;
                }

                if(!isString)
                {
                        if(number < 1 || number > 6)

                        {
                                System.out.println("Must be a number between 1 and "
+ board[0].length);
                        }
                        else
                        {
                                number -= 1;

                                int index = board.length - 1;

                                while(index >= 0)
                                {
                                        if(board[index][number] == '\u0000')
                                        {
                                                board[index][number] = player;
                                                return number;
                                        }

                                        index--;
                                }

                                System.out.println("Column full.");
                        }
                }

        }

          return 0;
    }
```

```
	/**
	 * Checks if the player, with the character on top on the received column, won
	 * the game or not. It will get the top move on that column, and check if there
	 * are 4 pieces in a row, in relation to that piece and from the same player.
	 * Returns true is yes, false is not.
	 *
	 * @param board The board
	 * @param col   The last played column
	 * @return True is that player won the game, or false if not.
	 */
	private static boolean lastPlayerWon(char[][] board, int col) {
		char player = '0';
		int row = 0;

		for(int rows = 0; rows < board.length; rows++)
		{
			if(board[rows][col] != '\u0000')
			{
				player = board[rows][col];
				row = rows;
				break;
			}
		}


		//Evaluate Row.
		int inARow = 0;
		int rowAux = row;

		while(rowAux < board.length)
		{
			//System.out.print("Row: " + rowAux);

			if(board[rowAux][col] == player)
			{
				inARow++;

				if(inARow > 3)
				{
					return true;
				}
			}
			else
			{
				inARow = 0;
			}

			//System.out.println(" | In A Row: " + inARow);
			rowAux++;
		}
```

```
	//Evaluate Col
	inARow = 0;
	int colAux = 0;

	while(colAux < board[row].length)
	{
		//System.out.print("Col: " + colAux);

		if(board[row][colAux] == player)
		{
			inARow++;

			if(inARow > 3)
			{
				return true;
			}
		}
		else
		{
			inARow = 0;
		}

		//System.out.println(" | In A Row: " + inARow);
		colAux++;
	}


	//Evaluate Diagonal - Top Left to Bottom Right
	inARow = 0;
	rowAux = row;
	colAux = col;

	//System.out.println("--- Diagonal Top Left to Bottom Right ---");

	//System.out.println("Initial Row: " + rowAux + " | Initial Col: " +
colAux);

	//Find Diagonal Beginning
	while(rowAux > 0 && colAux > 0)
	{
		rowAux--;
		colAux--;
	}

	//System.out.println("Final Row: " + rowAux + " | Final Col: " +
colAux);
```

```
	//Check Diagonal
	while(rowAux < board.length - 1 && colAux < board[rowAux].length - 1)
	{
		//System.out.print("Diagonal - Row: " + rowAux + " | Col: " +
colAux);

		if(board[rowAux][colAux] == player)
		{
			inARow++;

			if(inARow > 3)
			{
				return true;
			}
		}
		else
		{
			inARow = 0;
		}

		//System.out.println(" | In A Row: " + inARow);
		rowAux++;
		colAux++;
	}

	//Evaluate Diagonal - Top Right to Bottom Left
	inARow = 0;
	rowAux = row;
	colAux = col;

	//System.out.println("--- Diagonal Top Right to Bottom Left ---");

	//System.out.println("Initial Row: " + rowAux + " | Initial Col: " +
colAux);

	//Find Diagonal Beginning
	while(rowAux < board.length - 1 && colAux < board[rowAux].length - 1)
	{
		rowAux++;
		colAux++;
	}

	//System.out.println("Final Row: " + rowAux + " | Final Col: " +
colAux);
```

```
        //Check Diagonal
        while(rowAux > 0 && colAux > 0)
        {
              //System.out.print("Diagonal - Row: " + rowAux + " | Col: " +
colAux);

              if(board[rowAux][colAux] == player)
              {
                    inARow++;

                    if(inARow > 3)
                    {
                          return true;
                    }
              }
              else
              {
                    inARow = 0;
              }

              //System.out.println(" | In A Row: " + inARow);
              rowAux--;
              colAux--;
        }

          return false;
    }


     /**
     * Check if there are at least one free position on board.
     *
     * @param board The board
     * @return True if there is, at least, one free position on board
     */
    private static boolean existsFreePlaces(char[][] board) {

        for(int rows = 0; rows < board.length; rows++)
        {
              for(int cols = 0; cols < board[rows].length; cols++)
              {
                    if(board[rows][cols] == '\u0000')
                    {
                          return true;
                    }
              }
        }

          return false;
    }
```

```
    /**
      * Main method - this method should not be changed
      */
     public static void main(String[] args) {
         final int NCOLs = 7;
         final int NROWS = 6;

         // program variables
         Scanner keyboard = new Scanner(System.in);
         char[][] board = new char[NCOLs][NROWS];
         char winner = ' ';

         // show empty board
         showboard(board);

         // game cycle
         do {
             int col = play('A', board, keyboard);
             showboard(board);
             if (lastPlayerWon(board, col)) {
                 winner = 'A';
                 break;
             }
             if (!existsFreePlaces(board))
                 break;

             col = play('B', board, keyboard);
             showboard(board);
             if (lastPlayerWon(board, col)) {
                 winner = 'B';
                 break;
             }

         } while (existsFreePlaces(board));

        // show final result
         switch (winner) {
             case ' ':
                 System.out.println("We have a draw....");
                 break;
             case 'A':
                 System.out.println("Winner: Player A. Congratulations...");
                 break;
             case 'B':
                 System.out.println("Winner: Player B. Congratulations...");
                 break;
         }

         // close keyboard
         keyboard.close();
     }
}
```

1.3. P03WorkWithStrings

Neste programa existem apenas dois métodos. O método *compareStrings,* que recebe duas *strings* como argumentos e que devolve um inteiro, sendo este negativo ou positivo, e o *main* que, para vários exemplos, chama o *compareStrings*. O primeiro passo é identificar se uma ou ambas as *strings* são *null*. De seguida, é verificado o seu tamanho, seguindo-se a comparação por caracteres. Em cada uma destas comparações, é possível obter os valores finais deste método.

Código:

```
package tp1.pack1Revisoes;

public class P03WorkWithStrings {

    /**
     * Main, método de arranque da execução
     */
    public static void main(String[] args) {
        test_compareStrings(null, null); // result = 0
        test_compareStrings(null, ""); // result = -1
        test_compareStrings("", null); // result = 1
        test_compareStrings("a", ""); // result = 1
        test_compareStrings("", "a"); // result = -1
        test_compareStrings("a", "a"); // result = 0
        test_compareStrings("b", "a"); // result = 1
        test_compareStrings("a", "b"); // result = -1
        test_compareStrings("aa", "a"); // result = 2
        test_compareStrings("a", "aa"); // result = -2
        test_compareStrings("aa", "aa"); // result = 0
        test_compareStrings("ab", "aa"); // result = 2
        test_compareStrings("ab", "ab"); // result = 0
        test_compareStrings("abc", "abc"); // result = 0
        test_compareStrings("abc", "abd"); // result = -3
    }
```

```java
   /**
     * Este método recebe duas Strings s1 e s2 e procede à sua comparação,
     * devolvendo um valor positivo se s1 for maior que s2, negativo se ao
      * contrário e 0 se iguais. A comparação deve ser feita primeiro em
termos
       * lexicográficos caracter a caracter começando pelos caracteres de
menor
       * peso ou em segundo lugar em termos de número de caracteres. Se
diferentes
     * deve devolver o índice +1/-1 do caractere que faz a diferença. Ex.
       * s1="Bom", s2="Dia", deve devolver -1; s1="Boa", s2="Bom", deve
devolver
        * -3; s1="Bom", s2="Bo", deve devolver 3. Uma String a null é
considerada
     * menor que uma string não null.
     *
     * @param s1 string a comparar
     * @param s2 string a comparar
     * @return o resultado da comparação
     */
    private static int compareStrings(String s1, String s2) {

      // ---- Null Check
      if(s1 == null && s2 == null)
      {
            return 0;
      }
      else if(s1 == null)
      {
            return -1;
      }
      else if(s2 == null)
      {
            return 1;
      }
      // ---- Null Check End

      // ---- Length Check
      if(s1.length() == 0 && s2.length() == 0)
      {
            return 0;
      }
      else if(s1.length() == 0)
      {
            return -1;
      }
      else if(s2.length() == 0)
      {
            return 1;
      }
      // ---- Length Check End
```

```
        // ---- Character Check
        int smallerLength = 0;

        boolean foundDifference = false;
        int differentCharacterIndex = 0;

        if(s1.length() < s2.length())
        {
              smallerLength = s1.length();
        }
        else if(s1.length() > s2.length())
        {
              smallerLength = s2.length();
        }
        else if(s1.length() == s2.length())
        {
              smallerLength = s1.length();
        }

        for(int charIndex = 0; charIndex < smallerLength; charIndex++)
        {
              if(s1.charAt(charIndex) != s2.charAt(charIndex))
              {
                    //System.out.print("Difference in Char's (s1 " + s1 + " |
s2 " + s2 + "): " + charIndex);
                    foundDifference = true;
                    differentCharacterIndex = charIndex;
                    break;
              }
        }

        //Passed ALL Char's and didn't find any difference.
        if(!foundDifference)
        {
              if(s1.length() < s2.length())
              {
                    return -s2.length();
              }
              else if(s2.length() < s1.length())
              {
                    return s1.length();
              }
              else
              {
                    return 0;
              }
        }
```

```
        if(s1.charAt(differentCharacterIndex) < s2.charAt(differentCharacterIndex))
        {
            return -differentCharacterIndex - 1;
        }
        else if(s2.charAt(differentCharacterIndex) < s1.charAt(differentCharacterIndex))
        {
            return differentCharacterIndex + 1;
        }
        // ---- Character Check End

        return 0;
    }

    /**
     * Auxiliary method that call compareStrings with two strings
     */
    private static void test_compareStrings(String s1, String s2) {
        try {
            System.out.print("compareStrings (" + s1 + ", " + s2 + ") = ");
            int res = compareStrings(s1, s2);
            System.out.println(res);

        } catch (IllegalArgumentException e) {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

2. pack2Livros

2.1. Livro

No constructor da classe Livro, o título não pode ser *null* e não pode ter 0 caracteres. O número de páginas tem de ser positivo, assim como o preço. Os autores não podem possuir *nulls* e têm de conter, pelo menos, uma letra. Só podem conter letras e espaços. Depois disso remover os espaços extra e saber se existem repetições. *getTitulo*, *getNumPaginas* e *getPreco* devolvem os respectivos atributos. *getAutores* devolve uma cópia do *array* de autores. *validarNome* verifica se os nomes possuem apenas letras e espaços, assim como pelo menos uma letra. *validarNomes* usa o *validarNome* para cada índice do *array* recebido. *removeExtraSpaces* remove os espaços a mais em na *string* recebida. *haRepeticoes* verifica se existem repetições no *array* recebido. *contemAutor* verifica se o autor recebido é um autor desse Livro. *toString* imprime o conteúdo do Livro de forma clara. *print* adiciona um prefixo ao *toString*. *equals* compara o título do Livro recebido ao mesmo. E, por fim, o *main* serve para criação de diversos livros e testes dos métodos.

Código:

```
package tp1.pack2Livros;

/**
 * Classe que deverá suportar um livro
 */
public class Livro {

    // Título do livro
    private String titulo;

    // número de páginas
    private int numPaginas;

    // preço do livro
    private float preco;

    // array de autores, este array não deve ter nulls
    private String[] autores;
```

```java
 /**
      * Deve criar um novo livro com os dados recebidos. O título não deve ser
       * null nem vazio. O número de páginas não pode ser menor que 1. O preço não
       * pode ser negativo. O array de autores não deve conter nem nulls e deve
       * conter pelo menos um autor válido. Não pode haver repetições dos nomes
      * dos autores, considera-se os nomes sem os espaços extra (ver
      * removeExtraSpaces). Este método deve utilizar os métodos auxiliares
      * existentes. Em caso de nome inválido deve lançar uma excepção de
      * IllegalArgumentException com a indicação do erro ocorrido
      */
       public Livro(String titulo, int numPaginas, float preco, String[] autores) {

        // título
        if (titulo == null || titulo.length() == 0)
               throw new IllegalArgumentException("O titulo tem de ter pelo menos um caracter");
        this.titulo = titulo;

        //Numero de Paginas
        if(numPaginas < 1)
        {
             throw new IllegalArgumentException("O nº de páginas não pode ser negativo");
            //Acho que devia ficar "Tem de ter pelo menos uma página".
        }
        this.numPaginas = numPaginas;

        //Preco
        if(preco < 0)
        {
                  throw new IllegalArgumentException("O preco nao pode ser negativo");
        }
        this.preco = preco;


        //Autores
        if(!validarNomes(autores))
        {
             this.autores = autores;
             throw new IllegalArgumentException("Autores invalidos");
        }

        String[] autoresSemEspacos = new String[autores.length];

        for(int autorIndex = 0; autorIndex < autores.length; autorIndex++)
        {
             autoresSemEspacos[autorIndex] = removeExtraSpaces(autores[autorIndex]);
        }
```

```
        if(haRepeticoes(autoresSemEspacos))
        {
            throw new IllegalArgumentException("O array de autores contém
autores repetidos");
        }

        this.autores = autoresSemEspacos;
    }

    /**
     * Devolve o título do livro
     */
    public String getTitulo() {

        return titulo;
    }

    /**
     * Devolve o número de páginas do livro
     */
    public int getNumPaginas() {

        return numPaginas;
    }

     /**
     * Devolve o preço do livro
     */
    public float getPreco() {

        return preco;
    }
    /**
     * Devolve uma cópia do array de autores do livro
     */
    public String[] getAutores() {

        return autores.clone();
    }

    /**
     * Deve devolver true se o array conter apenas nomes válidos. Um nome é
       * válido se conter pelo menos uma letra (Character.isLetter) e só
conter
     * letras e espaços (Character.isWhitespace). Deve chamar validarNome.
     */
    public static boolean validarNomes(String[] nomes) {

      for(String nome : nomes)
      {
            if(!validarNome(nome))
            {
                  return false;
            }
      }

        return true;
    }
```

```
     /**
      * Um nome válido se não for null e não conter pelo menos uma letra
      * (Character.isLetter) e só conter letras e espaços
      * (Character.isWhitespace)
      */
    public static boolean validarNome(String nome) {

      if(nome == null)
      {
            return false;
      }

      for(int charIndex = 0; charIndex < nome.length(); charIndex++)
      {
            if(!Character.isLetter(nome.charAt(charIndex))                  &&
!Character.isWhitespace(nome.charAt(charIndex)))
                  {
                  return false;
                  }
      }

        return true;
    }
```

```java
/**
	 * Recebe um nome já previamente validado, ou seja só com letras ou
espaços.
	 * Deve devolver o mesmo nome mas sem espaços (utilizar trim e
	 * Character.isWhitespace) no início nem no fim e só com um espaço ' '
entre
	 * cada nome. Deve utilizar um StringBuilder para ir contendo o nome já
	 * corrigido
	 */
	public static String removeExtraSpaces(String nome) {

		StringBuilder builder = new StringBuilder();
		nome = nome.trim();

		int charIndex = 0;
		int whitespaceInARow = 0;

		while(charIndex < nome.length())
		{
			if(Character.isWhitespace(nome.charAt(charIndex)))
			{
				whitespaceInARow++;

				if(whitespaceInARow < 2)
				{
					builder.append(nome.charAt(charIndex));
				}
			}
			else
			{
				whitespaceInARow = 0;
				builder.append(nome.charAt(charIndex));
			}

			charIndex++;
		}

		return builder.toString();
	}

	/**
	 * Método que verifica se há elementos repetidos. O array recebido não
	 * contém nulls.
	 */
	public static boolean haRepeticoes(String[] elems) {

		for(int elemsIndex = 0; elemsIndex < elems.length; elemsIndex++)
		{
			for(int elemsCheckIndex = 0; elemsCheckIndex < elems.length;
elemsCheckIndex++)
			{
				if(elems[elemsIndex].equals(elems[elemsCheckIndex]) &&
elemsIndex != elemsCheckIndex)
				{
					return true;
				}
			}
		}

		return false;
	}
```

```
    /**
     * Devolve true se o autor recebido existe como autor do livro. O nome
     * recebido não contém espaços extra.
     */
    public boolean contemAutor(String autorNome) {

      for(String autor : autores)
      {
            if(autor.equals(autorNome))
            {
                  return true;
            }
      }

        return false;
    }

    /**
     * Devolve uma string com a informação do livro (ver outputs desejados)
     */
    public String toString() {

      String output = titulo + ", " + numPaginas + "p " + preco + " [";

      for(int autorIndex = 0; autorIndex < autores.length; autorIndex++)
      {
            if(autorIndex == autores.length - 1)
            {
                  output += autores[autorIndex] + "]";
            }
            else
            {
                  output += autores[autorIndex] + ", ";
            }
      }

        return output;
    }

    /**
     * Deve mostrar na consola a informação do livro precedida do prefixo
     */
    public void print(String prefix) {
        System.out.println(prefix + toString());
    }

    /**
      * O Livro recebido é igual se tiver o mesmo título que o título do
livro
     * corrente
     */
    public boolean equals(Livro l) {

      if(l.getTitulo().equals(titulo))
      {
            return true;
      }

        return false;
    }
```

```
    /**
     * main
     */
    public static void main(String[] args) {

        // constructor e toString
             Livro l = new Livro("Viagem aos Himalaias", 340, 12.3f, new
String[]{"João Mendonça", "Mário Andrade"});
        System.out.println("Livro -> " + l);
        l.print("");
        l.print("-> ");
        System.out.println();

        // contém autor
        String autorNome = "Mário Andrade";
           System.out.println("Livro com o autor " + autorNome + "? -> " +
l.contemAutor(autorNome));
        autorNome = "Mário Zambujal";
           System.out.println("Livro com o autor " + autorNome + "? -> " +
l.contemAutor(autorNome));
        System.out.println();

        // equals
        System.out.println("Livro: " + l);
        System.out.println("equals Livro: " + l);
        System.out.println(" -> " + l.equals(l));

             Livro l2 = new Livro("Viagem aos Himalaias", 100, 10.3f, new
String[]{"Vitor Záspara"});
        System.out.println("Livro: " + l);
        System.out.println("equals Livro: " + l2);
        System.out.println(" -> " + l.equals(l2));
        System.out.println();

        // testes que dão excepção - mostra-se a excepção

         // livro lx1
        System.out.println("Livro lx1: ");
        try {
               Livro lx1 = new Livro("Viagem aos Himalaias", -1, 12.3f, new
String[]{"João Mendonça", "Mário Andrade"});
            System.out.println("Livro lx1: " + lx1);
        } catch (IllegalArgumentException ex) {
            ex.printStackTrace();
        }
        System.out.println();

        // livro lx2
        System.out.println("Livro lx2: ");
        try {
            Livro lx2 = new Livro("Viagem aos Himalaias", 200, -12.3f,
                    new String[]{"João Mendonça", "Mário Andrade"});
            System.out.println("Livro lx2: " + lx2);
        } catch (IllegalArgumentException ex) {
            ex.printStackTrace();
        }
        System.out.println();
```

```
        // livro lx3
        System.out.println("Livro lx3: ");
        try {
                Livro lx3 = new Livro(null, 200, -12.3f, new String[]{"João
Mendonça", "Mário Andrade"});
            System.out.println("Livro lx3: " + lx3);
        } catch (IllegalArgumentException ex) {
            ex.printStackTrace();
        }
        System.out.println();

        // livro lx4
        System.out.println("Livro lx4: ");
        try {
            Livro lx4 = new Livro("Viagem aos Himalaias", 200, 12.3f,
                        new String[]{"João Mendonça", "Mário Andrade", "João
Mendonça"});
            System.out.println("Livro lx4: " + lx4);
        } catch (IllegalArgumentException ex) {
            ex.printStackTrace();
        }
    }
}
```

2.2. Coleccao

No constructor da classe Coleccao, o título não pode ser *null* e deve ter, pelo menos, um caractere. Deve existir pelo menos um editor e este também tem de ter pelo menos um caractere. *getTitulo* devolve o título da colecção e o *getNumPaginas* devolve a soma de todas as páginas dos Livros. *getPreco* devolve o preço da colecção que pode ter um desconto. *addLivro* verifica se é possível adicionar um Livro à colecção, se o Livro não é *null* e se não é repetido. *getIndexOfLivro* devolve o índice do Livro, caso exista, da colecção. *remLivro* devolve o Livro que se pretende remover, caso exista, e arranja o *array* de forma a não haver *nulls* entre os Livros. *getNumObrasFromPerson* devolve o número de obras da pessoa. *getLivrosComoAutor* devolve um *array* com os Livros dos quais a pessoa recebida como argumento, é autora. *toString* imprime o conteúdo da Colecção de forma clara. *getAutoresEditores* devolve os editores e os autores dos Livros da colecção. *mergeWithoutRepetition* une os arrays recebidos, removendo os índices repetidos. *equals* compara a colecção atual com a recebida, baseando-se no título e nos editores. *print* adiciona um prefixo ao *toString*. Por fim, o *main* serve para criação de diversas colecções, livros e testes dos métodos.

Código:

```
package tp1.pack2Livros;

import java.util.Arrays;

/**
 * Classe Colecca, deve conter a descrição de uma colecção, com título, seus
 * livros e editores
 */
public class Coleccao {

    // número máximo de obras de uma colecção
    private static int MAXOBRAS = 20;

    // prefixo usual
    public static final String GENERALPREFIX = "  ";

    // título da colecção
    private String titulo;

      // Array de livros, em que estas encontram-se sempre nos menores índices e
    // pela ordem de registo
    private Livro[] livros = new Livro[MAXOBRAS];
```

```
   // deverá conter sempre o número de livros na colecção
    private int numLivros = 0;

      // Editores, tem as mesmas condicionantes que array de autores na
classe
    // livro
    private String[] editores;

    /**
     * Construtor; o título tem de ter pelo menos um caracter que não seja
um
     * espaço (Character.isWhitespace); o array de editores devem ser pelo
menos
     * um e têm as mesmas restrições que os autores dos livros; o array de
     * livros deve conter os mesmos sempre nos menores índices
     */
    public Coleccao(String titulo, String[] editores) {
        // titulo
        if (titulo == null || titulo.length() == 0)
            throw new IllegalArgumentException(
                    "O titulo tem de ter pelo menos um caracter");

                for(int tituloIndex = 0; tituloIndex < titulo.length();
tituloIndex++)
        {
            if(!Character.isWhitespace(tituloIndex))
            {
                  break;
            }

            if(tituloIndex == titulo.length() - 1)
            {
                  throw new IllegalArgumentException("O titulo nao pode
conter apenas espacos");
            }
        }

        this.titulo = titulo;

         //Editores
        if(editores.length <= 0)
        {
            throw new IllegalArgumentException("Tem de existir, pelo menos,
um editor");
        }

        for(String editor : editores)
        {
            if (editor == null || editor.length() == 0)
                 throw new IllegalArgumentException("O editor tem de ter pelo
menos um caracter");
        }

        this.editores = editores;
    }
```

```
     /**
      *
      */
    public String getTitulo() {

        return titulo;
    }

    /**
     * Obtem o número total de páginas da colecção
     */
    public int getNumPaginas() {

      int numPaginas = 0;

      for(int index = 0; index < numLivros; index++)
      {
            numPaginas += livros[index].getNumPaginas();
      }

        return numPaginas;
    }

     /**
      * Devolve o preço da colecção tendo em conta que as colecções com 4 ou
mais
      * livros têm um desconto de 20% nos livros que custam pelo menos 10
euros e
      * que têm mais de 200 páginas
      */
    public float getPreco() {

      float preco = 0;

      if(numLivros < 4)
      {
            for(int index = 0; index < numLivros; index++)
            {
                  preco += livros[index].getPreco();
            }
      }
      else
      {
            for(int index = 0; index < numLivros; index++)
            {
                  Livro livro = livros[index];

                  if(livro.getPreco() >= 10 && livro.getNumPaginas() > 200)
                  {
                        preco += livro.getPreco() * 0.8f;
                  }
                  else
                  {
                        preco += livro.getPreco();
                  }
            }
      }

      return preco;
    }
```

```
     /**
      * Adiciona um livro se puder e este não seja null e a colecção não
ficar
      * com livros iguais. Deve utilzar o método getIndexOfLivro.
      */
    public boolean addLivro(Livro livro) {

      if(numLivros == livros.length - 1 || livro == null)
      {
            return false;
      }

      if(getIndexOfLivro(livro.getTitulo()) != -1)
      {
            return false;
      }

      livros[numLivros] = livro;
      numLivros++;

        return true;
    }

    /**
     * Devolve o index no array de livros onde estiver o livro com o nome
     * pretendido. Devolve -1 caso não o encontre
     */
    private int getIndexOfLivro(String titulo) {

      for(int livroIndex = 0; livroIndex < numLivros; livroIndex++)
      {
            if(livros[livroIndex].getTitulo().equals(titulo))
            {
                  return livroIndex;
            }
      }

        return -1;
    }
```

```java
 /**
       * Remove do array o livro com o título igual ao título recebido. Devolve o
     * livro removido ou null caso não tenha encontrado o livro. Deve-se
      * utilizar o método getIndexOfLivro. Recorda-se que os livros devem ocupar
      * sempre os menores índices, ou seja, não pode haver nulls entre os livros
     */
    public Livro remLivro(String titulo) {
      boolean exists = false;
      Livro livroARemover = null;
      int livroIndex = 0;

      for(int indexLivros = 0; indexLivros < numLivros; indexLivros++)
      {
            if(livros[indexLivros].getTitulo().equals(titulo))
            {
                  exists = true;
                  livroARemover = livros[indexLivros];
                  livroIndex = indexLivros;
            }
      }

      if(!exists)
      {
            return null;
      }

      Livro[] novosLivros = new Livro[numLivros - 1];
      int indexes = 0;

      for(int indexLivros = 0; indexLivros < numLivros; indexLivros++)
      {
            if(indexLivros != livroIndex)
            {
                  novosLivros[indexes] = livros[indexLivros];
                  indexes++;
            }
      }

      livros = novosLivros;
      numLivros--;

      return livroARemover;
      }
```

```
      /**
       * Devolve o nº de obras de uma pessoa. A colecção deve
contabilizar-se como
       * uma obra para os editores.
       */
      public int getNumObrasFromPerson(String autorEditor) {

        int numObras = 0;

        for(String editor : editores)
        {
          if(editor.equals(autorEditor))
          {
            numObras++;
            }
      }

      for(int indexLivro = 0; indexLivro < numLivros; indexLivro++)
      {
            for(String autor : livros[indexLivro].getAutores())
            {
                  if(autor.equals(autorEditor))
                  {
                        numObras++;
                  }
            }
      }

        return numObras;
    }

      /**
       * Devolver um novo array (sem nulls) com os livros de que a pessoa
recebida
     * é autor
     */
    public Livro[] getLivrosComoAutor(String autorNome) {

      Livro[] livrosComoAutor = new Livro[numLivros];
      int numLivrosComoAutor = 0;

      for(int livroIndex = 0; livroIndex < numLivros; livroIndex++)
      {
            if(livros[livroIndex].contemAutor(autorNome))
            {
                  livrosComoAutor[numLivrosComoAutor] = livros[livroIndex];
                  numLivrosComoAutor++;
            }
      }

      Livro[] livrosComoAutorFinal = new Livro[numLivrosComoAutor];

      for(int   livrosIndex   =   0;   livrosIndex   <   numLivrosComoAutor;
livrosIndex++)
      {
            livrosComoAutorFinal[livrosIndex]                                =
livrosComoAutor[livrosIndex];
      }

        return livrosComoAutorFinal;
    }
```

```
     /**
      * Deve devolver uma string compatível com os outputs desejados
      */
    public String toString() {

        String toBeReturned = "Colecção " + titulo + ", editores [ ";

        for(int index = 0; index < editores.length; index++)
        {
            if(index == editores.length - 1)
            {
                  toBeReturned += editores[index] + "], ";
            }
            else
            {
                  toBeReturned += editores[index] + ", ";
            }
        }

          toBeReturned += numLivros + " livros, " + getNumPaginas() + "p " +
getPreco();

        return toBeReturned;
    }

    /**
      * Deve devolver um array, sem nulls, com todos os autores e editores
      * existentes na colecção. O resultado não deve conter repetições. Deve
      * utilizar o método mergeWithoutRepetitions
      */
    public String[] getAutoresEditores() {

      String[] autoresEditores = editores.clone();

      for(int index = 0; index < numLivros; index++)
      {
            autoresEditores     =     mergeWithoutRepetitions(autoresEditores,
livros[index].getAutores());
      }

        return autoresEditores;
    }
```

```
 /**
       * Método que recebendo dois arrays sem repetições devolve um novo
array com
      * todos os elementos dos arrays recebidos mas sem repetições
      */
      private static String[] mergeWithoutRepetitions(String[] a1, String[]
a2) {

            int tamanhoFinal = a1.length;

            String[] naoRepetidos = new String[a2.length];
            int indexNaoRepetidos = 0;

            for(String texto : a2)
            {
                  boolean repetido = false;

                  for(String textoAComparar : a1)
                  {
                        if(texto.equals(textoAComparar))
                        {
                              repetido = true;
                        }
                  }

                  if(!repetido)
                  {
                        naoRepetidos[indexNaoRepetidos] = texto;
                        indexNaoRepetidos++;

                        tamanhoFinal++;
                  }
            }

            String[] arraySemRepeticoes = new String[tamanhoFinal];
            int indexSemRepeticoes = 0;

            for(int index = 0; index < a1.length; index++)
            {
                  arraySemRepeticoes[indexSemRepeticoes] = a1[index];
                  indexSemRepeticoes++;
            }

            for(int index = 0; index < indexNaoRepetidos; index++)
            {
                  arraySemRepeticoes[indexSemRepeticoes]                    =
naoRepetidos[index];
            }

        return arraySemRepeticoes;
    }
```

```
    /**
     * Devolve true caso a colecção recebida tenha o mesmo título e a mesma
      * lista de editores. Para verificar verificar se os editores são os
mesmos
     * devem utilizar o método mergeWithoutRepetitions
     */
    public boolean equals(Coleccao c) {

      if(!c.getTitulo().equals(titulo))
      {
            return false;
      }

      String[] merge = mergeWithoutRepetitions(editores, c.editores);

      if(merge.length     !=     editores.length     ||     merge.length     !=
c.editores.length)
      {
            return false;
      }

        return true;
    }

     /**
     * Mostra uma colecção segundo os outputs desejados
     */
    public void print(String prefix) {
        System.out.println(prefix + toString());

        for(int indexLivro = 0; indexLivro < numLivros; indexLivro++)
        {
            livros[indexLivro].print(" ");
        }
    }

    /**
     * main
     */
    public static void main(String[] args) {
        Livro l1 = new Livro("Viagem aos Himalaias", 340, 12.3f,
                new String[]{"João Mendonça", "Mário Andrade"});
        Livro l2 = new Livro("Viagem aos Pirinéus", 270, 11.5f,
                new String[]{"João Mendonça", "Júlio Pomar"});

        Coleccao c1 = new Coleccao("Primavera",
                new String[]{"João Mendonça", "Manuel Alfazema"});

        boolean res;

        res = c1.addLivro(l1);
        res = c1.addLivro(l2);
        System.out.println("c1 -> " + c1);
        c1.print("");
        System.out.println();
```

```
        // adicionar um livro com nome de outro já existente
        res = c1.addLivro(l2);
        System.out.println(
                "adição novamente de Viagem aos Pirinéus a c1 -> " + res);
        System.out.println("c1 -> " + c1);
        System.out.println();

        // get editores autores
        String[] ae = c1.getAutoresEditores();
                    System.out.println("Autores editores of c1 -> " +
Arrays.toString(ae));
        System.out.println();

        // getNumObrasFromPerson
        String nome = "João Mendonça";
        int n = c1.getNumObrasFromPerson(nome);
        System.out.println("N° de obras de " + nome + " -> " + n);
        System.out.println();

        // getLivrosComoAutor
        nome = "João Mendonça";
        Livro[] obras = c1.getLivrosComoAutor(nome);
        System.out
                                .println("Livros de " + nome + " -> " +
Arrays.toString(obras));
        System.out.println();

        // rem livro
        String nomeLivro = "Viagem aos Himalaias";
        Livro l = c1.remLivro(nomeLivro);
        System.out.println("Remoção de " + nomeLivro + " -> " + l);
        c1.print("");
        System.out.println();

        // equals
        Coleccao c2 = new Coleccao("Primavera",
                new String[]{"João Mendonça", "Manuel Alfazema"});
        boolean same = c1.equals(c2);
        System.out.println("c2:");
        c2.print("");
        System.out.println("c1.equals(c2) -> " + same);
        System.out.println();

        Coleccao c3 = new Coleccao("Primavera",
                new String[]{"João Mendonça"});
        same = c1.equals(c3);
        System.out.println("c3:");
        c3.print("");
        System.out.println("c1.equals(c3) -> " + same);
    }
}
```

3. pack3Coleccoes

3.1. Coleccao

Muitos dos métodos apresentados nesta classe são semelhantes aos métodos apresentados na classe Coleccao do segundo package. Dado isto, vou apenas clarificar as funções que apresentam mudanças e novos métodos.
getNumPaginas percorre também o *array* de coleccções para somar o número de páginas dessas. *getPreco* apresenta novos descontos e mudanças para abranger as colecções. *addColeccao* verifica se é possível adicionar uma colecção. *getIndexOfColeccao* devolve o índice da colecção recebida como argumento, caso esta exista. *remColeccao* devolve a colecção que se pretende remover e reorganizar o *array* das colecções, evitando *nulls* entre colecções. *getNumObrasFromPerson* abrange agora as sub-colecções, assim como o *getLivrosComoAutor* abrange os livros das sub-colecções. *getAutoresEditores* abrange sub-colecções e livros das mesmas. *mergeWithoutRepetitions* para *strings* e agora também para *Livros*. *string* agora também percorre sub-colecções.

Código:

```
package tp1.pack3Coleccoes;

import java.util.Arrays;

import tp1.pack2Livros.Livro;

/**
 * Classe Coleccao, deve conter a descrição de uma colecção, com título, os
seus
 * livros, colecções e editores
 */
public class Coleccao {
    // número máximo de obras de uma colecção
    private static int MAXOBRAS = 20;

    // prefixo usual
    public static final String GENERALPREFIX = "  ";

    // título da colecção
    private String titulo;

      // Array de livros, em que estas encontram-se sempre nos menores
índices e
    // pela ordem de registo
    private Livro[] livros = new Livro[MAXOBRAS];

    // deverá conter sempre o número de livros na colecção
    private int numLivros = 0;

    // array de colecções, estas devem ocupar sempre os menores índices
    private Coleccao[] coleccoes = new Coleccao[MAXOBRAS];

    // deverá conter sempre o número de colecções dentro da colecção
    private int numColeccoes = 0;
```

```
      // Editores, tem as mesmas condicionantes que array de autores na
classe
    // livro
    private String[] editores;

    /**
      * Construtor; o título tem de ter pelo menos um caracter que não seja
um
      * espaço (Character.isWhitespace); o array de editores devem ser pelo
menos
      * um e têm as mesmas restrições que os autores dos livros;
      */
    public Coleccao(String titulo, String[] editores) {
        // titulo
        if (titulo == null || titulo.length() == 0)
            throw new IllegalArgumentException(
                    "O titulo tem de ter pelo menos um caracter");
        this.titulo = titulo;

                for(int tituloIndex = 0; tituloIndex < titulo.length();
tituloIndex++)
        {
            if(!Character.isWhitespace(tituloIndex))
            {
                  break;
            }

            if(tituloIndex == titulo.length() - 1)
            {
                  throw new IllegalArgumentException("O titulo nao pode
conter apenas espacos");
            }
        }

        this.titulo = titulo;

         //Editores
        if(editores.length <= 0)
        {
            throw new IllegalArgumentException("Tem de existir, pelo menos,
um editor");
        }

        for(String editor : editores)
        {
            if (editor == null || editor.length() == 0)
                 throw new IllegalArgumentException("O editor tem de ter pelo
menos um caracter");
        }

        this.editores = editores;
    }
```

```
     /**
      *
      */
    public String getTitulo() {

        return titulo;
    }

     /**
      * Obtem o número total de páginas da colecção, páginas dos livros e
das
      * colecções
      */
    public int getNumPaginas() {

      int paginas = 0;

      for(int coleccoesIndex = 0; coleccoesIndex < numColeccoes;
coleccoesIndex++)
      {
            paginas += coleccoes[coleccoesIndex].getNumPaginas();
      }

      for(int livrosIndex = 0; livrosIndex < numLivros; livrosIndex++)
      {
            paginas += livros[livrosIndex].getNumPaginas();
      }

        return paginas;
    }
```

```
 /**
      * As colecções com mais de 5000 páginas nos seus livros directos têm
um
       * desconto de 20% nesses livros. As colecções em que o somatório de
páginas
      * das suas subcolecções directas seja igual ou superior ao quádruplo
do nº
       * de páginas da sua subcolecção directa com mais páginas deverão
aplicar um
     * desconto de 10% sobre os preços das suas subcolecções
     */
    public float getPreco() {

      float preco = 0;

      int numPaginas = 0;

      for(int index = 0; index < numLivros; index++)
      {
            numPaginas += livros[index].getNumPaginas();
            preco += livros[index].getPreco();
      }

      if(numPaginas > 5000)
      {
            preco *= 0.8f;
      }

      numPaginas = 0;
      int maximoNumeroDePaginas = 0;

      for(int index = 0; index < numColeccoes; index++)
      {
            if(coleccoes[index].getNumPaginas() > maximoNumeroDePaginas)
            {
                  maximoNumeroDePaginas = coleccoes[index].getNumPaginas();
            }
      }

      for(int index = 0; index < numColeccoes; index++)
      {
            numPaginas += coleccoes[index].getNumPaginas();
      }

      if(numPaginas >= 4 * maximoNumeroDePaginas)
      {
            for(int index = 0; index < numColeccoes; index++)
            {
                  preco += (coleccoes[index].getPreco() * 0.9);
            }
      }

      for(int index = 0; index < numColeccoes; index++)
            {
                  preco += coleccoes[index].getPreco();
            }

        return preco;
    }
```

```
       /**
        * Adiciona um livro à colecção se puder e este não seja null e a
colecção
        * não ficar com livros iguais ao nível imediato da colecção. Deve
utilzar o
     * método getIndexOfLivro e getIndexOfColeccao
     */
    public boolean addLivro(Livro livro) {

      if(numLivros == livros.length - 1 || livro == null)
      {
            return false;
      }

      if(getIndexOfLivro(livro.getTitulo()) != -1)
      {
            return false;
      }

      livros[numLivros] = livro;
      numLivros++;

        return true;
    }

      /**
     * Adiciona uma colecção à colecção se puder, esta não seja null e a
     * colecção não ficar com obras imediatas com títulos repetidos. Deve
     * utilizar o método getIndexOfLivro e getIndexOfColeccao
     */
    public boolean addColeccao(Coleccao col) {

      if(numColeccoes == coleccoes.length - 1 || col == null)
      {
            return false;
      }

      if(getIndexOfColeccao(col.getTitulo()) != -1)
      {
            return false;
      }

      coleccoes[numColeccoes] = col;
      numColeccoes++;

        return true;
    }
```

```
     /**
      * Devolve o index no array de livros onde estiver o livro com o nome
      * pretendido. Devolve -1 caso não o encontre
      */
    private int getIndexOfLivro(String titulo) {

      for(int index = 0; index < numLivros; index++)
      {
            if(livros[index].getTitulo().equals(titulo))
            {
                  return index;
            }
      }

        return -1;
    }

     /**
      * Devolve o index no array de colecções onde estiver a colecção com o
nome
      * pretendido. Devolve -1 caso não o encontre
      */
    private int getIndexOfColeccao(String titulo) {

      for(int index = 0; index < numColeccoes; index++)
      {
            if(coleccoes[index].getTitulo().equals(titulo))
            {
                  return index;
            }
      }

        return -1;
    }
```

```
      /**
       * Remove do array o livro com o título igual ao título recebido. Devolve o
      * livro removido ou null caso não tenha encontrado o livro. Deve-se
       * utilizar o método getIndexOfLivro. Recorda-se que os livros devem ocupar
       * sempre os menores índices, ou seja, não pode haver nulls entre os livros
      */
    public Livro remLivro(String titulo) {

      boolean exists = false;
      int indexLivro = 0;
      Livro livroARemover = null;

      for(int index = 0; index < numLivros; index++)
      {
            if(livros[index].getTitulo().equals(titulo))
            {
                  exists = true;
                  indexLivro = index;
                  livroARemover = livros[index];
            }
      }

      if(!exists)
      {
            return null;
      }

        Livro[] novosLivros = new Livro[numLivros - 1];
        int novosLivrosIndex = 0;

        for(int index = 0; index < numLivros; index++)
        {
            if(index != indexLivro)
            {
                  novosLivros[novosLivrosIndex] = livros[index];
                  novosLivrosIndex++;
            }
        }

        livros = novosLivros;
        numLivros--;

        return livroARemover;

    }
```

```java
/**
     * Remove do array de colecções a colecção com o título igual ao título
       * recebido. Devolve a colecção removida ou null caso não tenha
encontrado.
       * Deve-se utilizar o método getIndexOfColeccao. Recorda-se que as
colecções
       * devem ocupar sempre os menores índices, ou seja, não pode haver
nulls
     * entre elas
     */
    public Coleccao remColeccao(String titulo) {

      boolean exists = false;
      int indexColeccao = 0;
      Coleccao coleccaoARemover = null;

      for(int index = 0; index < numColeccoes; index++)
      {
            if(coleccoes[index].getTitulo().equals(titulo))
            {
                  exists = true;
                  indexColeccao = index;
                  coleccaoARemover = coleccoes[index];
            }
      }

      if(!exists)
      {
            return null;
      }

      Coleccao[] novasColeccoes = new Coleccao[numColeccoes - 1];
      int novasColeccoesIndex = 0;

      for(int index = 0; index < numColeccoes; index++)
      {
            if(index != indexColeccao)
            {
                  novasColeccoes[novasColeccoesIndex] = coleccoes[index];
                  novasColeccoesIndex++;
            }
      }


      coleccoes = novasColeccoes;
      numColeccoes--;

        return coleccaoARemover;
    }
```

```
/**
         * Devolve o nº de obras de uma pessoa. Cada colecção deve
contabilizar-se
     * como uma obra para os editores.
     */
    public int getNumObrasFromPerson(String autorEditor) {

      int numObras = 0;

      for(int index = 0; index < numLivros; index++)
      {
            for(String autor: livros[index].getAutores())
            {
                  if(autor.equals(autorEditor))
                  {
                        numObras++;
                  }
            }
      }

      if(this instanceof Coleccao)
      {
            for(String autor : this.editores)
            {
                  if(autor.equals(autorEditor))
                  {
                        numObras++;
                  }
            }
      }

      for(int index = 0; index < numColeccoes; index++)
      {
            numObras += coleccoes[index].getNumObrasFromPerson(autorEditor);
      }

        return numObras;
    }
```

```java
/**
	 * Devolver um novo array (sem nulls) com os livros de que a pessoa recebida
	 * é autor. Não deve conter repetições, para excluir as repetições devem
	 * utilizar o método mergeWithoutRepetitions
	 */
	public Livro[] getLivrosComoAutor(String autorNome) {

		Livro[] livrosDoAutorDaColeccao = new Livro[numLivros];
		int livrosDoAutorIndex = 0;

		for(int index = 0; index < numLivros; index++)
		{
			for(String autor : livros[index].getAutores())
			{
				if(autorNome.equals(autor))
				{
					livrosDoAutorDaColeccao[livrosDoAutorIndex] = livros[index];
					livrosDoAutorIndex++;
				}
			}
		}

		Livro[] livrosDoAutor = new Livro[livrosDoAutorIndex];

		for(int index = 0; index < livrosDoAutorIndex; index++)
		{
			livrosDoAutor[index] = livrosDoAutorDaColeccao[index];
		}

		for(int index = 0; index < numColeccoes; index++)
		{
			livrosDoAutor = mergeWithoutRepetitions(livrosDoAutor, coleccoes[index].getLivrosComoAutor(autorNome));
		}

		return livrosDoAutor;
	}
```

```
/**
     * Deve devolver uma string compatível com os outputs desejados
     */
    public String toString() {

            String toBeReturned = "Colecção " + titulo + ", editores [ ";

        for(int index = 0; index < editores.length; index++)
        {
            if(index == editores.length - 1)
            {
                  toBeReturned += editores[index] + "], ";
            }
            else
            {
                  toBeReturned += editores[index] + ", ";
            }
        }

          toBeReturned += numLivros + " livros, " + getNumPaginas() + "p " +
getPreco();

        return toBeReturned;
    }

     /**
     * Deve devolver um array, sem nulls, com todos os autores e editores
     * existentes na colecção. O resultado não deve conter repetições. Deve
     * utilizar o método mergeWithoutRepetitions
     */
    public String[] getAutoresEditores() {

      String[] merged = editores.clone();

      for(int index = 0; index < numLivros; index++)
      {
            merged              =              mergeWithoutRepetitions(merged,
livros[index].getAutores());
      }

      for(int index = 0; index < numColeccoes; index++)
      {
            merged              =              mergeWithoutRepetitions(merged,
coleccoes[index].getAutoresEditores());
      }

        return merged;
    }
```

```
/**
       * Método que recebendo dois arrays sem repetições devolve um novo
array com
      * todos os elementos dos arrays recebidos mas sem repetições
      */
      private static String[] mergeWithoutRepetitions(String[] a1, String[]
a2) {

       int tamanhoFinal = a1.length;

             String[] naoRepetidos = new String[a2.length];
             int indexNaoRepetidos = 0;

             for(String texto : a2)
             {
                   boolean repetido = false;

                   for(String textoAComparar : a1)
                   {
                         if(texto.equals(textoAComparar))
                         {
                               repetido = true;
                         }
                   }

                   if(!repetido)
                   {
                         naoRepetidos[indexNaoRepetidos] = texto;
                         indexNaoRepetidos++;

                         tamanhoFinal++;
                   }
             }

             String[] arraySemRepeticoes = new String[tamanhoFinal];
             int indexSemRepeticoes = 0;

             for(int index = 0; index < a1.length; index++)
             {
                   arraySemRepeticoes[indexSemRepeticoes] = a1[index];
                   indexSemRepeticoes++;
             }

             for(int index = 0; index < indexNaoRepetidos; index++)
             {
                   arraySemRepeticoes[indexSemRepeticoes]                    =
naoRepetidos[index];
             }

        return arraySemRepeticoes;

    }
```

```
/**
     * Método idêntico ao método anterior mas agora com arrays de livros
     */
    private static Livro[] mergeWithoutRepetitions(Livro[] a1, Livro[] a2) {

      int tamanhoFinal = a1.length;

            Livro[] naoRepetidos = new Livro[a2.length];
            int indexNaoRepetidos = 0;

            for(Livro livro : a2)
            {
                  boolean repetido = false;

                  for(Livro livroAComparar : a1)
                  {
                        if(livro != null)
                        {
                              if(livro.equals(livroAComparar))
                              {
                                    repetido = true;
                              }
                        }
                  }

                  if(!repetido)
                  {
                        naoRepetidos[indexNaoRepetidos] = livro;
                        indexNaoRepetidos++;

                        tamanhoFinal++;
                  }
            }

            Livro[] arraySemRepeticoes = new Livro[tamanhoFinal];
            int indexSemRepeticoes = 0;

            for(int index = 0; index < a1.length; index++)
            {
                  if(a1[index] != null)
                  {
                        arraySemRepeticoes[indexSemRepeticoes] = a1[index];
                        indexSemRepeticoes++;
                  }
            }

            for(int index = 0; index < indexNaoRepetidos; index++)
            {
                  arraySemRepeticoes[indexSemRepeticoes]                    =
naoRepetidos[index];
                  indexSemRepeticoes++;
            }

        return arraySemRepeticoes;
    }
```

```
      /**
      * Devolve true caso a colecção recebida tenha o mesmo título e a mesma
       * lista de editores. Para verificar verificar se os editores são os
mesmos
      * devem utilizar o método mergeWithoutRepetitions
      */
    public boolean equals(Coleccao c) {

      if(!c.getTitulo().equals(titulo))
      {
            return false;
      }

      String[] merge = mergeWithoutRepetitions(editores, c.editores);

      if(merge.length     !=     editores.length     ||     merge.length     !=
c.editores.length)
      {
            return false;
      }

        return true;
    }

     /**
      * Mostra uma colecção segundo os outputs desejados
      */
    public void print(String prefix) {

      System.out.println(prefix + toString());

      for(int index = 0; index < numLivros; index++)
      {
            livros[index].print(prefix + " ");
      }

      for(int index = 0; index < numColeccoes; index++)
      {
            coleccoes[index].print(" ");
      }
    }
```

```
    /**
     * main
     */
    public static void main(String[] args) {
        Livro l1 = new Livro("Viagem aos Himalaias", 340, 12.3f,
                new String[]{"João Mendonça", "Mário Andrade"});
        Livro l2 = new Livro("Viagem aos Pirinéus", 270, 11.5f,
                new String[]{"João Mendonça", "Júlio Pomar"});

        Coleccao c1 = new Coleccao("Primavera",
                new String[]{"João Mendonça", "Manuel Alfazema"});

        boolean res;

        res = c1.addLivro(l1);
        res = c1.addLivro(l2);
        System.out.println("c1 -> " + c1);
        c1.print("");
        System.out.println();

        // adicionar um livro com nome de outro já existente
        res = c1.addLivro(l2);
        System.out.println(
                "adição novamente de Viagem aos Pirinéus a c1 -> " + res);
        System.out.println("c1 -> " + c1);
        System.out.println();

        // Outra colecção
        Livro l21 = new Livro("Viagem aos Himalaias 2", 340, 12.3f,
                new String[]{"João Mendonça", "Mário Andrade"});
        Livro l22 = new Livro("Viagem aos Pirinéus 2", 270, 11.5f,
                new String[]{"João Mendonça", "Júlio Pomar"});

        Coleccao cx2 = new Coleccao("Outono",
                new String[]{"João Mendonça", "Manuel Antunes"});
        cx2.addLivro(l21);
        cx2.addLivro(l22);
        System.out.println("cx2 -> " + cx2);
        cx2.print("");
        System.out.println();

        // adicioná-la a c1
        c1.addColeccao(cx2);
        System.out.println("c1 após adição da colecção cx2 -> " + c1);
        c1.print("");
        System.out.println();

        // get editores autores
        String[] ae = c1.getAutoresEditores();
                    System.out.println("Autores  editores  of  c1  ->  "  +
Arrays.toString(ae));
        System.out.println();

        // getNumObrasFromPerson
        String nome = "João Mendonça";
        int n = c1.getNumObrasFromPerson(nome);
        System.out.println("N° de obras de " + nome + " -> " + n);
        System.out.println();
```

```
        // getLivrosComoAutor
        nome = "João Mendonça";
        Livro[] obras = c1.getLivrosComoAutor(nome);
        System.out
                                .println("Livros de " + nome + " -> " +
Arrays.toString(obras));
        System.out.println();

         // rem livro
        String nomeLivro = "Viagem aos Himalaias";
        Livro l = c1.remLivro(nomeLivro);
        System.out.println("Remoção de " + nomeLivro + " -> " + l);
        c1.print("");
        System.out.println();
    }
}
```